Microprocessors and Microsystems 76 (2020) 103086

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A method for embedding a computer vision application into a wearable device

Elias T. Silva Jr*, Fausto Sampaio, Lucas C. da Silva, David S. Medeiros, Gustavo P. Correia

Computer Science Department, Federal Institute of Education, Science and Technology of Ceará. Av Treze de Maio, 2081. Fortaleza, CE, Brazil

ARTICLE INFO

Article history:
Received 21 August 2019
Revised 12 February 2020
Accepted 9 March 2020
Available online 10 March 2020

Keywords:
Embedded systems
Design space exploration
Computer vision
Pattern detection
Low-power wearable applications

ABSTRACT

Pattern classification applications can be found everywhere, especially the ones that use computer vision. What makes them difficult to embed is the fact that they often require a lot of computational resources. Embedded computer vision has been applied in many contexts, such as industrial or home automation, robotics, and assistive technologies. This work performs a design space exploration in an image classification system and embeds a computer vision application into a minimum resource platform, targeting wearable devices. The feature extractor and the classifier are evaluated for memory usage and computation time. A method is proposed to optimize such characteristics, leading to a reduction of over 99% in computation time and 92% in memory usage, with respect to a standard implementation. Experimental results in an ARM Cortex-M platform showed a total classification time of 0.3 s, maintaining the same accuracy as in the simulation performed. Furthermore, less than 20 KB of data memory was required, which is the most limited resource available in low-cost and low-power microcontrollers. The target application, used for the experimental evaluation, is a crosswalk detector used to help visually impaired persons.

1. Introduction

In order to increase efficiency and productivity in a variety of contexts, computer vision systems have been developed and used to classify patterns and detect objects in products and processes. However, many times these systems demand high processing capacity and large data memory, which imposes an extra challenge for embedding those applications. The target application platform, often has resource constraints, such as low memory sizes and low CPU performance, which must be taken into account in the design stages.

The development of machine learning solutions to practical problems on workstations, where there are no limitations of hardware resources, is common but then the exploration of the real constraints to embed these machine learning applications is often left as a future work.

An interesting subset of these applications is targeted to fulfill the necessities of visually impaired persons, and the device is usually a wearable device placed on the user's body. These wearable devices have certain restrictions, as they require lightweight and long life batteries. This work aims to create a wearable computer vision device for visually impaired persons.

This work presents a design space exploration on a pattern classifier made of a GLCM (Gray Level Co-occurrence Matrix) combined with a SVM (Support Vector Machine) to collaborate with this device. This combination has proven to be effective for many texture identification applications [1–3].

The entire process was tested on a workstation and on an embedded platform, where a final implementation was assessed. The main contribution of this work is to evaluate and reduce the memory required in the classifier system so that it fits into a low power microcontroller platform, preserving the accuracy obtained in workstations with much larger resources.

This paper is organized as follows: Section 2 describes some of the state-of-the-art-systems that present related works as well as different attempts to embed computer vision systems. Section 3 describes the exploration of this work and results obtained using a workstation. Section 4 shows the results of the classifier in an embedded platform. Section 5 presents the conclusion of the paper and discusses the results obtained.

2. Related works

The first part of this Section presents other works that investigated embedded computer vision systems, especially those based

* Corresponding author.
E-mail addresses: elias@ifce.edu.br (E.T. Silva Jr), faustos@ppgcc.ifce.edu.br (F. Sampaio), lucas.costa@lit.ifce.edu.br (L.C. da Silva), david.silvamm@gmail.com (D.S. Medeiros), gustavo-pinheiroc@outlook.com (G.P. Correia).

https://doi.org/10.1016/j.micpro.2020.103086
0141-9331/

on SVM classifiers and GLCMs as the feature extractor. The second part of this section (2.2) introduces some works that applied machine learning to assistive technology as an example of computer vision to detect textures.

2.1. Embedded computer vision applications

Many researchers have used SVM in various applications, but few have attempted to embed this classifier. In [4] a method was proposed to recognize people in aerial images obtained from Small Unmanned Aerial Vehicles using Pattern Recognition Systems applied to image recognition. Some machine learning classifiers were tested, including SVM together with the Histogram of Oriented Gradients (HOG) as the feature extractor. The embedded platform for evaluation was a Raspberry Pi 2 Model B v1.1, which is a computer with many resources and demands too much power for a wearable device. This platform size is very common to validate embedded computer vision proposals [5,6]. In [7] the same experimental approach was used. However, that work only explored performance aspects, leaving memory usage investigations as an open question.

The work of [8] presented a study to classify and identify the diseases of mango leaves for Indian agriculture. The SVM classifier was implemented in the OpenCV library. The use of a hardware platform to embed the entire system was left as a future work.

The SVM, along with the GLCM, was used in [3] in order to implement an indoors navigation system for a mobile robot, using a topological map. In the application, the entire classification task was performed by a remote microcomputer, which communicated with the robot by radio frequency.

Other investigations did not evaluate their proposals in a real embedded platform or did not consider the costs in the embedded device.

Several published studies have used a co-occurrence matrix (CM) for the extraction of features to classify the images. However, few of them went beyond validation in simulators. In addition, there were no feasibility studies on this technique combined with a classifier.

On considering the feature extractor for the classifier, the authors in [9] state that the main disadvantage of using GLCM is that it requires a long computing time, especially for very large images. Thus, the author proposed an FPGA-based architecture to calculate the co-occurrence matrix and extract 6 features. With the same objective in mind, other works [10–12] have proposed other architectures, based on FPGA. These solutions converge to the use of specific hardware in order to reduce computation time, and causing most of them to suppress some of the GLCM's features but without justifying their choices. Most of them did not associate the GLCM with a classifier, so they did not evaluate the classifier hit rate.

Starting from a different hypothesis, this research carried out design space exploration in memory usage and computation time, and concurrently evaluating the correctness of predictions compared to the reference implementations. Additionally, this work targeted a wearable device, and therefore needs to reduce cost and power; so we innovated using a microcontroller-based platform running an SVM classifier for computer vision.

In [13], a Cortex-M4 microcontroller is also used to perform image processing. The paper describes the embedded implementation of a bio-inspired vision system to avoid collisions, using a previously proposed neural network. The 32-bit micro control unit used was the STM32F407 that is clocked at 168 MHz and offers a total static random access memory (SRAM) of 192 KB. The authors stated that their model is ideal for embedded platforms and their device consumed a total SRAM of 100 KB. They did many real-world tests to evaluate their micro-robot implementations. However, they did not evaluate the (eventual) accuracy reduction caused by their modifications in the original algorithm.

2.2. Proposals to help visually impaired persons

There are many computer vision applications that use pattern classification and are suitable for this investigation. In this paper, an automatic Crosswalk Detection application was chosen.

In the literature, several strategies have been used to detect crosswalks. In [14] multi-resolution morphological image processing was used to detect pedestrian tracks and traffic lights. Using (640 $\times$ 360) pixel images, an accuracy of 98.33% was obtained for the crosswalks, and 89.47% accuracy for detecting pedestrian traffic signals. The algorithms were implemented in a portable device, based on an Intel Atom Z3735F working at 1.33 GHz, and reaching a computation time of 218.1 ms. No memory evaluation was performed. The author did not use embedded platforms nor machine learning to detect patterns. Based on the chosen platform, it can be inferred that the solution uses a lot of power.

Poggi [15] used a CNN (Convolutional Neural Network) with a 2-layer MLP (Multi Layer Perceptron) architecture to detect crosswalks. The device used images captured by a 3D camera and classified them into 5 different classes: vertical, horizontal, diagonal left, diagonal right and others (no crosswalks). The proposed method achieved an accuracy between 88% and 94% for the 5 trained classes, with 200ms computing time in a 1.7 GHz Exynos 4412 Prime device. This is another high-performance CPU implementation, which is more focused on the classifier accuracy than on power saving.

An SVM classifier using Local binary patterns (LBP), GLCM and a combination of both as characteristic extractors was used in [16]. The images were obtained from a satellite and the computation was performed by a smartphone. The proposed method obtained an accuracy of 96.6% using LBP, 90.9% using LBP + GLCM and 90.3% using GLCM. Other methods using deep learning such as AlexNet, VGG and GoogLeNet are applied in satellite images to detect crosswalks in [17], obtaining an average accuracy of 97.11%.

Usually, the investigation with an embedded platform is left for future works, or the chosen platforms for validation has "abundant" computational and energy resources, like smartphones, and the resources consumed by the device are not evaluated. The present investigation is more focused on very restricted platforms that could be easily worn by visually impaired persons, also proposing strategies for computational resources optimization.

3. Explorations on processing time and memory usage

The embedded platforms that offer lower power consumption, also have memory and performance restrictions. Therefore, it is important to investigate (and optimize) the amount of those resources consumed by the algorithms targeting a wearable computer vision device.

3.1. Database

The image dataset used in this work consists of a set of 600 images of crosswalks with dimensions of (1280 $\times$ 720) pixels. There were four classes, with 150 images for each class in the dataset, which is publicly available in [18]. The crosswalks images were obtained at various angles, in horizontal and vertical plans. Some of the crosswalks were faded as can be seen in Fig. 1. The four classes in Fig. 1 are defined as: (1) crosswalk on the right, (2) crosswalk on the left, (3) crosswalk straight ahead, and (4) no crosswalk.

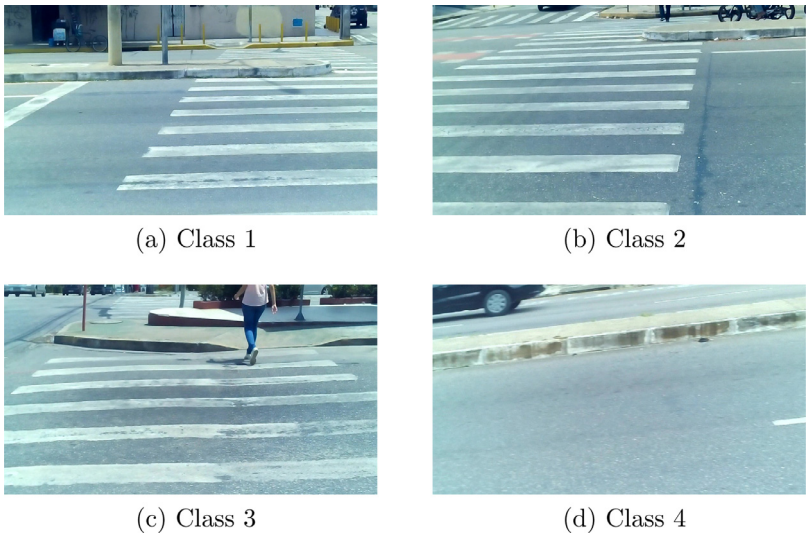

Fig. 1. Examples of classes.

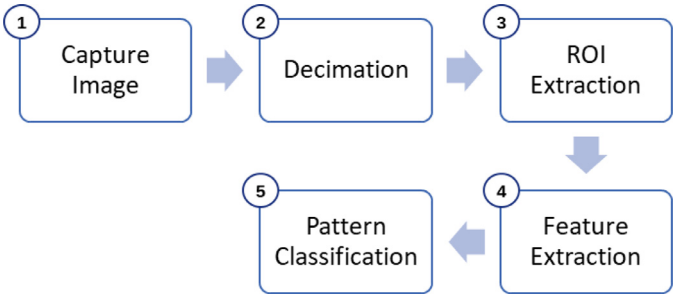

Fig. 2. Image classification steps.

3.2. Pattern recognition process

Fig. 2 illustrates the flow for a typical image classification process. After capturing the image (*Step 1*) a reduction in its resolution (*Step 2*) can be done and then the extraction of the ROI (Region Of Interest) is performed (*Step 3*).

In order to reduce the size and eliminate any redundancy of the data, features or characteristics are extracted and then used to represent the image (*Step 4*). The feature vectors (extracted from the image) are used in the training of a pattern classifier [19,20]. The method of feature extraction used in this work consisted of representing textures with second order statistics using a Gray Level Co-occurrence Matrix (GLCM) [21]. A Co-occurrence Matrix (CM) describes the amount of combinations of gray levels in an image with a certain direction and distance between the neighboring pixels. The size of this matrix (N_g) is equivalent to the amount of gray levels considered in the image.

The pattern classification process (*Step 5*) follows the feature extraction Step. For this work an SVM classifier was used. SVMs (Support Vector Machines) are classifiers that are based on the statistical learning theory, which takes into account the structural error minimization, calculated for the training vectors [22], and not only the minimization of the mean squared error (MSE). An SVM was chosen for the following reasons: it performs well in various computer vision applications [3,23,24]; and there are not many works, in the literature, related to embedding this classifier.

In order to train the SVM classifier, the image set of each class was randomly divided into the following proportions: 100 images (around 67%) for the training set and 50 for the test set. Altogether, 50 independent training and test runs were performed using the RBF (Radial Basis Function) *kernel*, with C and σ detected automatically by the grid-search.

A workstation (hereafter called WS1) was used to perform the initial evaluations of the classifier, before the entire process was deployed onto the embedded platform. The WS1 was an Intel Core i5 - 1.6 GHz, with 256 KB of L2 Cache per Core, 3 MB of L3 Cache, 8 GB of RAM - running macOS version 10.13.6, with the openCV software version 2.4.13.2 and in Python language. The classification process on the embedded system is explained in Section 4.

3.3. Computation time exploration

This Section is dedicated to explore properties in the pattern recognition process that can influence computation time. Special attention is given to the feature extractor.

3.3.1. Evaluation of GLCM features

The GLCM [21] [25] [26] provides a set of 24 statistical measures, which can be extracted from a single image. Table 1 lists all of these measures.

Initially, all 24 features were extracted from all 600 images of the crosswalks. Column Feature Set 1 (FS1) in Table 2 presents the accuracy of the SVM classifier for each class using all 24 features extracted from the images in their original size (1280 $\times$ 720). The accuracy and standard deviation values come from 50 runs.

Using 24 features gives a good average accuracy, around 91%, as shown in Table 2, column FS1. However, considering the goal is to minimize the size of the target embedded system, it does not seem to be the best solution. Thus, two other approaches were applied in reducing the number of features:

- Feature selection based on correlation (CFS).
- Feature computation study.

3.3.2. Feature selection based on correlation

CFS (Correlation-based Feature Selection) [27] is an algorithm that evaluates the performance of a subset of features by considering the individual predictive capacity of each feature along with the degree of redundancy between them. The expectation is to reduce the number of features used by the SVM classifier, as well as reducing the computational cost for both: feature extraction and classification. Among the 24 feature, the CFS selected 3: Sum Entropy, IMC I, Difference Mean.

Table 1
GLCM's Features.

Reference	#	Features
Haralick et al. (1973)[21]	14	Angular Second Moment (ASM), Contrast, Correlation, Sum of Squares, Sum Average, Inverse Difference Moment (IDM), Sum Variance, Sum Entropy, Entropy, Difference Variance, Difference Entropy, Information Measures of Correlation I (IMC I), Information Measures of Correl. II (IMC II), Maximal Correlation Coefficient (MCC).
Soh and Tsatsoulis (1999)[25]	6	Homogeneity, Autocorrelation, Dissimilarity, Cluster Shade, Cluster Prominence, Maximum Probability.
Wang et al. (2010)[26]	4	Sum Mean, Cluster Tendency, Difference Mean, Inertia.
Total	**24**	-

Table 2
Classifiers Accuracy for Each Feature Set with Original Image Size.

Class	Feature Set		
	FS1 (24 features)	FS2 (3 features)	FS3 (10 features)
1	91.54 ± 1.50	91.72 ± 2.02	91.65 ± 1.57
2	93.06 ± 1.62	92.62 ± 1.66	92.77 ± 1.85
3	88.77 ± 2.05	87.27 ± 2.11	89.15 ± 1.83
4	88.47 ± 1.90	88.23 ± 1.89	88.73 ± 1.68
Average	90.46 ± 1.23	89.96 ± 1.39	90.58 ± 1.21

The accuracy of the classifier using the set of features obtained by the CFS is shown in Table 2, column FS2.

Analyzing the results, the mean accuracy of the classifier was under 90%. Based on the accuracy, the CFS did not performed very well for this pattern recognition application. However, the reduction in the feature vectors will affect the computation time positively.

3.3.3. Feature selection based on calculation complexity

The order of growth of the running time of an algorithm allows the relative performance of alternative algorithms to be compared [28]. The complexity study is concerned with how the running time of an algorithm increases with the size of the input n. The notations that are often used to describe the processing time of an algorithm are called asymptotic notations. An algorithm that is asymptotically more efficient will be the best choice for all but very small inputs [28].

The asymptotic notation Θ was used to describe the running time of the algorithms of all 24 features, considering the worst case performances. The value of the input n is the size of the Co-occurrence Matrix, and is necessary to go through the entire matrix to extract a feature.

In the Table 3 all 24 features are presented according to its time complexity, based on the equations from [29]. Analyzing all values for the feature calculations, the lowest time complexity found was $\Theta(n^2)$.

Due to its low complexity, the 10 features in the $\Theta(n^2)$ group were selected for a new classifier accuracy analysis and named FS3 in Table 2. The accuracy results for the FS3 are nearly the same as those obtained by FS1. The accuracy of FS3 is close to FS1 because the features that were eliminated offered a negligible contribution to the classification. Moreover, those eliminated features tend to confuse the classifier.

However, the main advantage of the FS3 is in its smaller time usage. Note that some features selected by the CFS, like *IMC I*, have a computation time of $\Theta(2n^2)$, greater than $\Theta(n^2)$, which can be very time-consuming in a limited resources platform.

3.4. Memory usage exploration

This Section is dedicated to exploring properties in the pattern recognition process that can impact memory consumption. Three aspects are considered: The size of the image to be classified, the number of gray levels (a GLCM property), and the space complexity of the feature extraction.

3.4.1. Input image size exploration

In order to verify the impact that the image size causes on the classifier accuracy, tests were done with different image sizes. Starting from the original image size (1280 × 720), 99 new datasets were created, scaling down to 12 × 7.

The decimation technique was used to resize the original images, which produced an approximation of the sequence that would have been obtained by sampling the signal at a lower density [30]. In this work, a signal is equivalent to an image and the samples correspond to the image pixels.

To illustrate the image decimation, an input image I (6 × 8), after a decimation of $M = 3$, will result in an image J (2 × 3). The higher the value of M, the smaller the resultant image. In this example, only lines 1 and 4 of the input appear in the output image, as well as columns 1, 4, and 7. For the experiments made, M ranges from 1 to 100.

Fig. 3 presents examples of the crosswalk images, belonging to class 3 (crosswalk ahead), resized with the decimation technique.

Fig. 4 shows the classifier accuracy for different values of M, using only the FS3 set of features. As the image size decreases to around $M = 20$ (64 × 24 pixels), the accuracy decreases smoothly, with no significant losses in quality. However, the results show that using smaller size images (greater M) leads the classifier to make many mistakes. The images passed through a low-pass filter before decimation, although images from cameras usually have a minimum amount of high frequencies, and the filter does not significantly affect the accuracy. The experiment shown in Fig. 4 was also done without the filter, with almost the same results.

Table 4 shows the accuracy for some resized images, always using the FS3 features. To produce Table 4, filtered images were used, while in Table 2 the images were not filtered. It is possible to evaluate the filtering effect on accuracy by comparing the results for 1280 × 720 images in Table 4 with the results for the same resolution in Table 2 (FS3). The only difference between them is the low-pass filter. It can be seen that the difference is negligible, considering the standard deviation.

The results from Table 4 and Fig. 4 indicate that image resolution impacts the classifier accuracy but only when the image size is significantly reduced does it seriously degrade the accuracy. This occurs because the feature extractor adopted here (GLCM) is based on textures, and a small (or medium) image-size reduction preserves its texture.

In order to offer a significant reduction in memory usage while maintaining good accuracy, a criterion was established to select the image size. Fig. 5 illustrates the adopted approach. First of all, in order to select the acceptable results from each sample rate, the top 25% (third quartile) of the accuracy results are taken. In Fig. 5, this range is delimited by the dashes while the point indicates the Median. According to Fig. 4, higher image-size implies better accuracy. So, a quality range is set up, based on the worst accuracy of the three best results. This is shown in Fig. 5 by a red line, which is called the poor-quality line. Starting from the larger images, the smaller image-size to be accepted will be the one that precedes the first to have its median below the poor-quality line. This is a very conservative strategy, but the intention is to preserve

(a) $M = 1$ (b) $M = 17$ (c) $M = 27$

Fig. 3. Examples of a resized image from class 3.

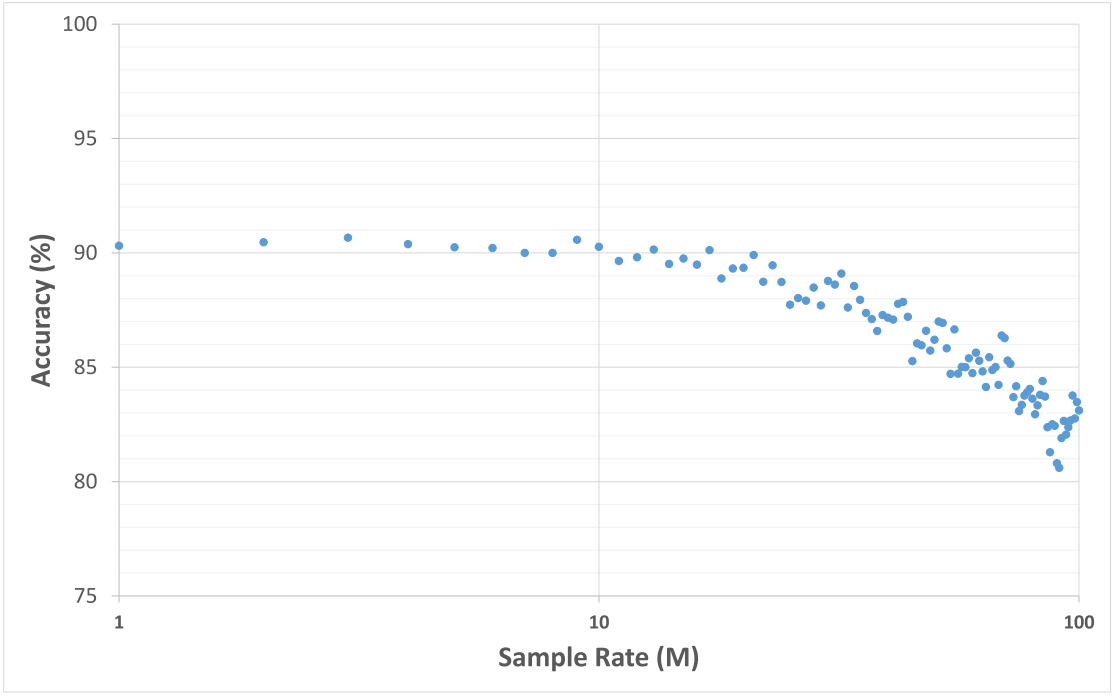

Fig. 4. SVM classifier accuracy for resized images using $1 \leq M \leq 100$.

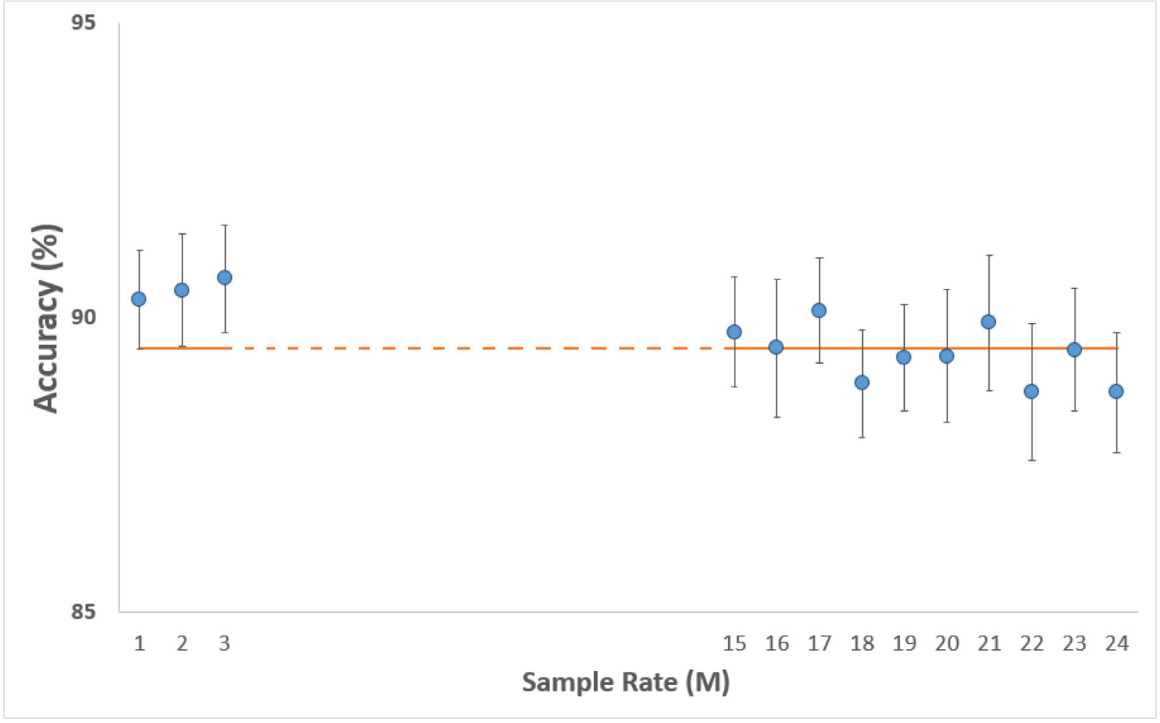

Fig. 5. Sampling deviation of the accuracy for some values of M.

Table 3
List of Features Grouped by Time Complexity.

Running Time	Features
$\Theta(n^2)$	ASM, Contrast, IDM, Entropy, Homogeneity, Sum Mean, Maximum Probability, Dissimilarity, Difference Mean, Autocorrelation.
$\Theta(n^2+n)$	Correlation, Inertia.
$\Theta(n^2+2n)$	Sum Average, Sum Entropy, Difference Variance, Difference Entropy.
$\Theta(n^2+4n-2)$	Sum Variance.
$\Theta(2n^2)$	Sum of Squares, IMC I,IMC II, MCC, Cluster Tendency, Cluster Shade, Cluster Prominence.

Table 4
Classifier Accuracy for Some Resized Images.

Class	M=1 (1280 × 720)	M=17 (76 × 43)	M=27 (48 × 27)
1	91.63 ± 1.81	91.15 ± 1.71	90.93 ± 1.77
2	92.89 ± 1.45	92.76 ± 1.68	88.86 ± 2.10
3	88.34 ± 1.59	88.36 ± 1.67	84.38 ± 2.35
4	88.38 ± 1.87	88.21 ± 1.95	87.47 ± 1.75
Average	90.31 ± 1.11	90.12 ± 1.18	87.91 ± 1.33

the quality of the classifier before reducing computational resource usage. For this particular dataset, the chosen sample rate (M) is 17, leading to a resolution of (76 × 43). Fig. 4 shows that there is a "better" sample rate (21) that performs very similar to that of 17. However, many of its results are under the red line, which does not happen with M=17. Moreover, keeping the tendency of Fig. 5 in mind, it is reasonable to assume that a higher image resolution will lead to a better classifier under the present operating conditions.

A reduction in image size influences both memory and computing time. So, in order to compare the computational effort to calculate the three feature sets, measurements of time spent for various image sizes were performed on the WS1 workstation and presented in Table 5. A hundred time-measurements were made for the whole process and the average was taken. According to Fig. 2, 'CM Matrix Creation' and 'Feature Extraction' are part of Step 4 and this includes a normalization. Steps 1 to 3 are not considered for time measurements.

Table 5 points out the following: (1) Creating the CM is very dependent on the size of the image. Therefore, there is a significant gain of time when using smaller images. (2) Feature extraction is independent of the image size, and only depends on the number of features. The results confirm the important reduction in time provided by the FS3 set. (3) The time spent in classification (SVM) is much less important, therefore it is justifiable to invest the optimization efforts on CM creation (image size) and feature extraction (based on complexity).

A slight variation in SVM computation time can occur as the resolution changes. Depending on the features, the SVM training process is forced to select a larger number of Support Vectors to be able to classify correctly. Therefore, the classifier might need more computation time.

3.4.2. Co-occurrence matrix evaluation

All the experiments performed so far have used a CM that considers 256 levels of gray (N_g), which are found in 8-bit resolution images. However, knowing that the amount of gray levels corresponds to the size of this matrix, a new experiment was proposed to compare results for different values of N_g. In this study, the CM was reduced, based on the number of bits, resulting in matrices proportional to the image resolution (number of bits or N). For example, for $N = 8$ and $N = 7$, the size of the CM will result, respectively, in 256X256 and 128X128.

Another classifier accuracy evaluation was performed but this time changing the number of gray levels. The graph in Fig. 6 presents the classifier accuracies for the following configuration:

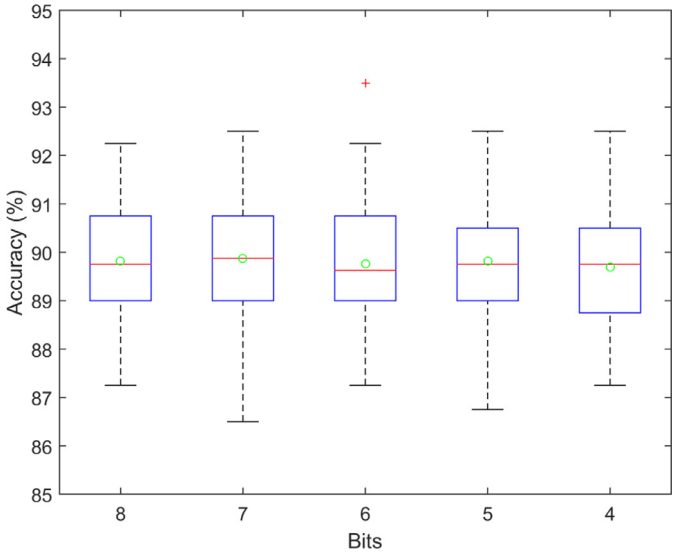

Fig. 6. Classifier accuracy for resized Co-occurrence Matrix.

resized images with $M = 17 (76 \times 43)$, FS3 feature set, and CM reduction using $N = \{8, 7, 6, 5, 4\}$, 50 training runs for each value of N.

The results in Fig. 6 indicate that a moderated reduction in the co-occurrence matrix does not cause any significant change to the classifier accuracy. Therefore, this new approach creates an opportunity for a strong reduction in the memory footprint. With this application, it is possible to use CM with dimensions of 64 × 64 ($N = 6$), for example, maintaining an average accuracy close to 90%.

The size of the Co-occurrence Matrix affects the time to calculate the features, as shown before in Table 3. Consequently, an extra gain in time is expected, besides the reduction in memory. In order to demonstrate that, Table 6 presents measurements of time spent considering some different gray levels and using the image size 76 × 43 ($M = 17$). The evaluation was executed on the WS1 workstation, taking the average time of 100 measurements. As expected, the time to calculate the features was benefited by the reduction of N_g. The CM creation time was not affected since it depends only on the input image size, where the CM is just the result of the process.

An image size of 76 × 43, and feature set FS2 (as shown in Table 5) could be used and this would result in a computing time of 17.7 seconds. Using the proposed approach ($N = 6$), the time would be 33.9 ms; which is a reduction of 99.8%.

3.4.3. Space complexity for feature extraction

Following that same approach presented in Section 3.3.3, a study of spatial complexity was made, in order to evaluate the memory occupancy for each feature calculation. Table 7 presents the results of the study for each feature calculation, grouped in crescent order, where n is the co-occurrence matrix size. The group of features that has a spatial complexity $\Theta(1)$, does not consume any extra memory (besides the co-occurrence matrix size). Feature

Table 5
Execution Time as a function of the image size and the Feature set, measured on the WS1 (in milliseconds).

Feature Set	Step	M=1 (1280x720)	M=17 (76x43)	M=27 (48x27)
FS1	CM Matrix Creation	1,313.955	5.393	2.184
	Feature Extraction	21,276.455	22,265.318	21,282.924
	Classification (SVM)	0.020	0.020	0.019
-	**Total**	**22,590.431**	**22,270.731**	**21,285.129**
FS2	CM Matrix Creation	1,376.806	5.067	2.183
	Feature Extraction	17,713.619	17,718.529	17,712.199
	Classification (SVM)	0.018	0.019	0.035
-	**Total**	**19,090.444**	**17,723.616**	**17,714.419**
FS3	CM Matrix Creation	1,274.454	4.872	2.268
	Feature Extraction	452.870	454.451	452.828
	Classification (SVM)	0.020	0.019	0.019
-	**Total**	**1,727.345**	**459.343**	**454.953**

Table 6
Execution Time as a function of the CM size, measured on the WS1 (in milliseconds) using FS3.

Step	8*bits* (256 × 256)	7*bits* (128 × 128)	6*bits* (64 × 64)	5*bits* (32 × 32)	4*bits* (16 × 16)
CM Creation	4.864	4.565	4.568	4.913	4.832
Feature Extraction	454.687	113.796	29.367	7.876	1.991
Classification (SVM)	0.019	0.019	0.023	0.023	0.021
Total (ms)	**459.571**	**118.381**	**33.959**	**12.814**	**6.836**

Table 7
List of features Grouped by Space Complexity.

Memory Cost	Features
$\Theta(1)$	ASM, Contrast, Sum of Squares, IDM, Entropy, Homogeneity, Sum Mean, Maximum Probability, Cluster Tendency, Cluster Shade, Cluster Prominence, Dissimilarity, Difference Mean, Autocorrelation, Inertia.
$\Theta(n)$	Difference Variance, Difference Entropy.
$\Theta(2n-1)$	Sum Average, Sum Entropy.
$\Theta(2n)$	Correlation, Sum Variance, IMC I, IMC II, MCC.

extractors that have a spatial complexity $\Theta(n)$ create an array of size (n) and so on.

Tables 3 and 7 demonstrate that the group containing the features with time complexity $\Theta(n^2)$ is included in the set of features with spatial complexity $\Theta(1)$. Thus, the 10 features with shorter computing time (FS3) also have the smaller spatial complexity.

4. Classifier evaluation on a 32-bit microcontroller platform

This Section describes the implementation of the image classifier into an embedded platform. Measurements of performance and memory footprint are presented and discussed as well as the embedded classifier accuracy.

4.1. Description of prototyping and testing platform

Since this is going to be a wearable application, it is important to evaluate the final results considering a low energy platform for small sized, low weight devices. Therefore, a limited resources microcontroller was chosen to embed this computer vision application.

A Tiva C Series TM4C123G Launchpad board was used as the embedded platform, which uses TM4C123GH6PM, an ARM Cortex-M4 based microcontroller. The TM4C123GH6PM has an 80 MHz clock speed, 256 KB Flash (program memory) and 32 KB SRAM (data memory). Additionally, the microcontroller power consumption is 148 mW with all peripherals active, running at 80 MHz. Although, it is suitable for wearable devices, some additional efforts should be made for better results.

The *Energia* 1.6.10E18 IDE was adopted to develop the application for the board, using C++ Language.

4.2. Replicating the pattern classifier on the microcontroller

The OpenCV was chosen as the tool to train the SVM due to its abundant documentation. The model was generated using the library version 2.4.13.2. The SVM classifier was trained and validated on the WS1 workstation using the following parameters: RBF kernel with $C = 12.5$ and $\sigma = 0.5$, FS3 set of features, 64 gray levels, image resolution of 76 × 43. A total of 50 training runs were performed.

The Co-occurrence Matrix (CM) creation and the algorithms to extract the 10 features were implemented in C++ based on the expressions provided in the literature [21,25,26,29].

Porting the SVM classifier to the embedded platform followed 2 steps: (1) Exporting the model generated by the training (in OpenCV), and (2) Inserting the model in the embedded classifier.

4.2.1. Exporting the OpenCV model generated by the training

The OpenCV offers a function that generates a text file with the SVM model. The file contains: the kernel type, the sigma value (called gamma), and a list containing the decision functions with their respective bias, alphas, and support vectors. The OpenCV SVM adopts the one-vs-one approach [31] that generates 6 binary classifiers (decision functions) for a 4-class problem, plus a voting system. The classifier with the best average accuracy was chosen to be embedded, which required 245 support vectors in the training process.

4.2.2. Inserting the model in the embedded classifier

The SVM classification code for the embedded platform was written in C++, based on the OpenCV references and source code on GitHub [32]. No optimization was introduced in this part. As can be seen in the previous results, such as in Table 6, the computation time of the SVM has a low impact on the total time. The

Table 8
Confusion matrix of the embedded implementation.

Class	1	2	3	4
1	119	2	19	10
2	0	126	12	12
3	14	9	114	13
4	10	12	9	119

Table 9
Performance of the embedded SVM classifier.

Class	Metrics	
	Acc(%)	$F1-Score$
1	90.83	81.23
2	92.17	84.28
3	87.33	75.00
4	89.00	78.29
Average	**89.83**	**79.70**

model generated by the OpenCV is included in the *Energia* project via a header file and is set to be implemented in the read-only memory (Flash).

4.3. Evaluating the embedded classifier

The validation of the pattern classification process was made by sending images to the board memory from a workstation through serial communications.

A script (on the workstation) sends images to the board that classifies them and sends the results back. The entire image set was sent to the board for evaluation, one by one. The class predicted by SVM as well as the features calculated by the board were captured on a file in the workstation. Knowing the expected class (provided by the OpenCV execution), the script compares it to the board output, counting a hit if the outputs match. The tests indicated 99.67% correctness; however, 2 samples were misclassified, compared to the workstation results and this was associated to the numerical precision of the microcontroller.

Table 8 shows the confusion matrix obtained by the embedded classifier when exposed to the whole data set. It is almost the same result as obtained by the WS1.

The performance of the embedded classifier can also be measured by its accuracy. Table 9 presents accuracy and $F1-Score$ for each class and the average. This classifier was evaluated using the whole dataset, which also contains some samples used in its training process. Despite this, the results are slightly worse than those in Table 4. As recommended in Section 3.4.2, the embedded version adopted a 64×64 co-occurrence matrix, while the experiment in Table 4 used a 256×256 CM. This information reduction can cause a small reduction in the accuracy of the classifier. Additionally, the data precision of the microcontroller is less than the workstation, contributing to some classifier errors.

The GLCM features calculated using the embedded system were also compared to the expected ones, obtained from the workstation. The maximum difference was 1.8150×10^{-4} in absolute values, which is an acceptable difference, knowing the features are normalized to the range 0 to 1.

In order to measure computation time, the classifier was made to run continuously getting images from the serial port and classifying them repeatedly. A GPIO pin is set to high at the beginning of the SVM computation and set to low at the end. Likewise, other GPIO pins are enabled for the CM (co-occurrence matrix) creation and the Feature Extraction. An oscilloscope (Tektronix MDO4034) captures the signals and calculates the average time spent at a high level for 100 repetitions. The results are presented in Table 10, where the last column on the right represents the percentage of each process.

Table 10
Time consumed (in milliseconds) on the embedded platform.

CM Creation	0.99	0.31%
Feature Extraction	306.40	94.60%
Classification (SVM)	16.48	5.09%
Total	**323.87**	100.00%

Table 11
Memory Usage (in bytes) on the embedded platform.

Flash	Decision Functions (6)	4,560	17.66%
	Support Vectors (245)	9,800	37.94%
	Code	11,468	44.40%
Total	-	**25,828**	100.00%
RAM	Image size (76×43)	3,268	15.72%
	Co-occurrence Matrix (64×64)	16,384	78.83%
	Features (10)	40	0.19%
	Other Variables	1,092	5.26%
Total	**-**	**20,784**	100.00%

Table 10 shows that the feature extraction was the most expensive operation, taking up more than 94% of the total time. This was known from the design space exploration steps explained in the previous Sections. The times for data retrieval and reporting of the results via the serial port were not included in the measures. The time to generate CM includes the ROI calculation, which uses a threshold algorithm that can easily be performed in the same loop.

The time to execute the low-pass filter in a (1280×720) image was also measured in the embedded platform, taking 0.94 ms. When it is necessary to use a filter before the image decimation there is no significant impact on the total computation time.

The Tiva C board, as many other microcontroller systems, has memory limitations, especially data memory. So, the immutable data must be placed in the FLASH memory, leaving the RAM to the mutable.

Table 11 details memory usage, extracted from the map file, generated by the linker. The last column on the right represents the percentage for each memory usage.

The SVM decision functions are stored in a 6-elements-array since 6 functions are required for a 4-class problem using one-vs-one strategy. Each array element (a struct) contains 4 fields: (*i*) the bias for the decision function (1 float); (*ii*) the number of support vectors needed for that function (1 integer); (*iii*) the indexes for each support vector used (94 integers); (*iv*) the alphas for each support vector in that function (94 floats). Note that not all decision functions use 94 support vectors, but the struct must be capable of storing the maximum number of indexes needed for one of them, which is 94 for this classifier. That is also the reason for storing the support vectors indexes for each decision function in the struct itself.

Table 11 demonstrates that the resizing of the CM plays a fundamental role to make this application feasible in a microcontroller platform because CM used 79% of the RAM. The final reduction surpasses 92% since the original CM size (256×256) would require 266,544 bytes to hold the application data.

The number of Support Vectors could be an issue to embed SVM classifiers, impacting memory usage and performance. However, they can be allocated in flash memory, which usually has plenty of space available. On the other hand, the time spent by the SVM decision process is not very important, as shown in Table 10. The Decision Functions also require a lot of memory for a non-binary classifier, especially when a one-vs-one approach is used. However, again, these data go to the flash memory.

Table 12
Estimated cost and power consumption for some embedded platforms.

	Cortex-M		RaspBerry	FPGA	
	F407	TM4C	Pi 3	Spartan7	Virtex 6/7
Power (mW)	175	180	800	-	70 to 950
Cost (USD)	19.9	13.5	35.0	99.0	1,390.00

Finally, a comparison was made among the different platforms proposed in the literature for embedded computer vision applications based on GLCM. Table 12 presents power consumption and cost for three different platforms, FPGA, Raspberry Pi (ARM Cortex-A), and ARM Cortex-M. Power estimation for the processors is easily extracted from their datasheets. On the other hand, power consumption in FPGA is highly dependent on the implementation. Experimental studies [33] for the Xilinx Virtex-II Pro-XC2VP30 indicate values from 70 mW to 950 mW. Those results depend on the implemented logic and how it was described.

Below, Table 12 presents general measurements in order to provide a simple comparison. Cortex-M is a microcontroller and its main characteristics are low power and low cost, targeting energy-efficient embedded devices. F407 is the microcontroller used in [13], although they do not use GLCM. The Cortex-A family provides high-performance processors, most of them being multi-core. The FPGA platforms are more focused on high-performance hardware implementations. They can provide a high number of operations per watt [34], and are more useful when high performance computation is required.

5. Conclusions

This work carried out a design space exploration for an image classification system. The implementation combines a GLCM (Gray Level Co-occurrence Matrix) and an SVM (Support Vector Machine), enabling the embedding of a computer vision application into a minimum resource platform. Considering that memory is the most limited resource in such platforms, the main contribution of this work is the implementation and evaluation of a reduced computational time and reduced memory usage of a computer vision system that fits into a low-power microcontroller platform, preserving the accuracy obtained in workstations which have much greater resources.

An empirical study was conducted using crosswalk detection, targeting a wearable application for the visually impaired.

A process to move an SVM predictor to a microcontroller platform is described, and ways to embed machine learning applications are discussed.

Two main qualities were explored to evaluate the consumption of computational resources: Memory usage and Execution time. Memory investigations showed results in 3 characteristics: Input image size, Number of gray levels, and Space complexity of the feature calculation. The effect of reducing the memory on the computation time was also evaluated.

The investigation focused on memory usage; however, the three characteristics investigated also resulted in benefits to the execution time, as stated in the results.

The experimental results indicate that it is not necessary to use the largest resolution available; a smaller resolution can still obtain good accuracy. A procedure was proposed to select that resolution.

The feature selection criteria based on time complexity proposed in [7] proved to be efficient not only in execution time but also in memory usage.

The number of gray levels (N_g) significantly impacted the GLCM memory usage due the creation of an intermediate matrix (size $N_g \times N_g$). The reduction of this matrix plays an essential role in making an application feasible in a microcontroller because it is usually the largest data structure to be deployed. The benefits include cost and power usage.

The main effort in optimization was in the feature extraction process, based on GLCM. In fact, the results showed that 94% of the classification time is spent in that process. The SVM classifier is complex, but the feature extraction is far more so.

The SVM memory footprint depends on two aspects: the number of Support Vectors (SV) and the number of Decision Functions (DF), both defined by the training process. Previous works usually neglected the latter because SVM is a binary classifier. However, applications with more classes tend to use more data for decision functions.

The memory used by the SVM (SV and DF) should be non-volatile (read-only) because the results deployed in the embedded system are generated by the training; and that data is not modified at runtime. The only exception is the feature vector used as input data. On the other hand, the memory required by the GLCM is basically RAM: the input image and Co-occurrence Matrix, beside the calculated features.

Wearable applications must be implemented on low-power platforms, and computational vision tends to become more and more common in such applications. This paper offers tips on how a computer vision application can be deployed on low-power and low-cost platforms.

Declaration of Competing Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgment

The authors would like to thank the sponsorship from FUNCAP and CAPES via grant no.05/2014 FUNCAP/CAPES.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.micpro.2020.103086.

References

[1] M. Talibi Alaoui, A. Sbihi, Texture classification based on co-occurrence matrix and neuro-morphological approach, in: A. Petrosino (Ed.), Image Analysis and Processing – ICIAP 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 510–521.
[2] A. Dasgupta, S. Grimaldi, R. Ramsankaran, J.P. Walker, Optimized glcm-based texture features for improved sar-based flood mapping, in: 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2017, pp. 3258–3261, doi:10.1109/IGARSS.2017.8127692.
[3] L.B. Marinho, J.S. Almeida, J.W.M. Souza, V.H.C. Albuquerque, P.P.R. Filho, A novel mobile robot localization approach based on topological maps using classification with reject option in omnidirectional images, Expert Syst. Appl. 72 (2017) 1–17, doi:10.1016/j.eswa.2016.12.007.
[4] D.C.D. Oliveira, M.A. Wehrmeister, Towards real-time people recognition on aerial imagery using convolutional neural networks, in: 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), IEEE, York, UK, 2016, pp. 27–34, doi:10.1109/ISORC.2016.14.
[5] R. Azarmehr, R. Laganiere, W.-S. Lee, C. Xu, D. Laroche, Real-time embedded age and gender classification in unconstrained video, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2015, pp. 56–64, doi:10.1109/cvprw.2015.7301367.
[6] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, J. Jang, Real-time driver drowsiness detection for embedded system using model compression of deep neural networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2017, pp. 438–445, doi:10.1109/cvprw.2017.59.
[7] F. Sampaio, L.C. da Silva, P.P.R. Filho, E.T. da Silva Jr, Reducing computational costs of an embedded classifier to determine leather quality, in: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), IEEE, 2017, pp. 211–216, doi:10.1109/sbesc.2017.36.

[8] J. Sethupathy, S. Veni, Opencv based disease identification of mango leaves, Int. J. Eng. Technol.Sci. Res. - IJETSR 8 (5) (2016) 1990–1998.
[9] L. Siéler, C. Tanougast, A. Bouridane, A scalable and embedded FPGA architecture for efficient computation of grey level co-occurrence matrices and haralick textures features, Microprocess. Microsyst. 34 (1) (2010) 14–24, doi:10.1016/j.micpro.2009.11.001.
[10] M. Tahir, A. Bouridane, F. Kurugollu, A. Amira, Accelerating the computation of GLCM and haralick texture features on reconfigurable hardware, in: 2004 International Conference on Image Processing. ICIP '04., IEEE, 2004, pp. 2857–2860Vol. 5, doi:10.1109/icip.2004.1421708.
[11] S. López-Estrada, R. Cumplido, Decision tree based FPGA-architecture for texture sea state classification, in: 2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006), IEEE, 2006, pp. 1–7, doi:10.1109/RECONF.2006.307770.
[12] D. Iakovidis, D. Maroulis, D. Bariamis, FPGA architecture for fast parallel computation of co-occurrence matrices, Microprocess. Microsyst. 31 (2) (2007) 160–165, doi:10.1016/j.micpro.2006.02.013.
[13] C. Hu, F. Arvin, C. Xiong, S. Yue, Bio-inspired embedded vision system for autonomous micro-robots: the lgmd case, IEEE Trans. Cognit. Dev.Syst. 9 (3) (2017) 241–254, doi:10.1109/TCDS.2016.2574624.
[14] K. Romic, I. Galic, H. Leventic, K. Nenadic, Real-time multiresolution crosswalk detection with walk light recognition for the blind, Adv. Electr. Comput. Eng. 18 (1) (2018) 11–20, doi:10.4316/aece.2018.01002.
[15] M. Poggi, L. Nanni, S. Mattoccia, Crosswalk recognition through point-cloud processing and deep-learning suited to a wearable mobility aid for the visually impaired, in: New Trends in Image Analysis and Processing – ICIAP 2015 Workshops, Springer International Publishing, 2015, pp. 282–289, doi:10.1007/978-3-319-23222-5_35.
[16] M.C. Ghilardi, J.J. Junior, I. Manssour, Crosswalk localization from low resolution satellite images to assist visually impaired people, IEEE Comput. Graph. Appl. 38 (1) (2018) 30–46, doi:10.1109/mcg.2016.50.
[17] R.F. Berriel, A.T. Lopes, A.F. de Souza, T. Oliveira-Santos, Deep learning-based large-scale automatic satellite crosswalk classification, IEEE Geosci. Remote Sens. Lett. 14 (9) (2017) 1513–1517, doi:10.1109/lgrs.2017.2719863.
[18] D.S. Medeiros, crosswalk-dataset, 2019, doi:10.34740/kaggle/dsv/846700.
[19] C.M. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer-Verlag, New York, 2006.
[20] G. Sinha, B.C. Patel, Medical Image Processing: Concepts and Applications, PHI Learning Private Limited, Delhi, 2014.
[21] R.M. Haralick, K. Shanmugam, I. Dinstein, Textural features for image classification, IEEE Trans. Syst. Man Cybern. SMC-3 (6) (1973) 610–621, doi:10.1109/tsmc.1973.4309314.
[22] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[23] W.P. Amorim, H. Pistori, M.A.C. Jacinto, A comparative analysis of attribute reduction algorithms applied to wet-blue leather defects classification, Brazilian Symp. Comput. Graph. Image Process. (2009).
[24] C. Lin, A support vector machine embedded weed identification system, University of Illinois at Urbana-Champaign, 2009 Master thesis.
[25] L.-K. Soh, C. Tsatsoulis, Texture analysis of SAR sea ice imagery using gray level co-occurrence matrices, IEEE Trans. Geosci. Remote Sens. 37 (2) (1999) 780–795, doi:10.1109/36.752194.
[26] H. Wang, X.-H. Guo, Z.-W. Jia, H.-K. Li, Z.-G. Liang, K.-C. Li, Q. He, Multilevel binomial logistic prediction model for malignant pulmonary nodules based on texture features of CT image, Eur. J. Radiol. 74 (1) (2010) 124–129, doi:10.1016/j.ejrad.2009.01.024.
[27] M.A. Hall, Correlation-based Feature Subset Selection for Machine Learning, University of Waikato, Hamilton, New Zealand, 1998 Ph.D. thesis.
[28] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3, The MIT Press Cambridge, Massachusetts, London, England, 2009.
[29] dGB Earth Sciences, Texture directional: a multi-trace attribute that returns textural information based on a statistical texture classification, 2015.
[30] R.G. Lyons, Understanding Digital Signal Processing, third, Prentice-Hall, Nova Jersey, 2010.
[31] K.-B. Duan, S.S. Keerthi, Which is the best multiclass SVM method? an empirical study, in: Multiple Classifier Systems, Springer Berlin Heidelberg, 2005, pp. 278–285, doi:10.1007/11494683_28.
[32] G. Bradski, The OpenCV library, Dr. Dobb's J. Softw. Tools (2000).
[33] D. Meidanis, K. Georgopoulos, I. Papaefstathiou, Fpga power consumption measurements and estimations under different implementation parameters, in: 2011 International Conference on Field-Programmable Technology, 2011, pp. 1–6, doi:10.1109/FPT.2011.6132694.
[34] P. Marwedel, Embedded System Design, second, Springer-Verlag, Berlin, Heidelberg, 2011, doi:10.1007/978-94-007-0257-8.

Elias Teodoro da Silva Junior received the BS degree in Electrical Engineering (Federal University of Ceará, Brazil), in 1991, Msc Degree in Electrical Engineering (Federal University of Sta. Catarina, Brazil), in 1994 and the Ph.D Degree in Computer Engineering (Federal University of Rio Grande do Sul, Brazil), in 2008. Currently, he is a Professor in the Computer Science Department, at Federal Institute of Education, Science and Technology of Ceará, Brazil. His research interests include embedded systems, machine learning and digital signal processing applications, and Internet of Things.

Fausto Sampaio received the Master's degree (2017) and Bachelor's degree (2014) in Computer Science at Federal Institute of Education, Science and Technology of Ceará. He has technical courses in Software Development (2007), Industrial Automation (2009) and Informatics (2010) at Federal Institute of Education, Science and Technology of Ceará. He is currently Systems Analyst at Federal University of Ceará. He has experience in Computer Science, with emphasis on Embedded Systems and Software Engineering, working mainly on the following topics: microcontrollers, robotics, pattern recognition, computer vision, system development and modeling.

Lucas Costa da Silva received the Bachelor's degree (2018) in Computer Engineering at Federal Institute of Education, Science and Technology of Ceará. He has technical courses in Software Development (2011) at Professional School Luiza de Teodoro Vieira. He is currently an M.Sc. candidate in Computer Science at Federal Institute of Education, Science and Technology of Ceará. His current research interests include Embedded Systems, IoT, Intelligent Computing, Neural Networks, Computer Vision and pattern recognition.

David Silva Medeiros has Bachelor's degree in Computer Engineering (2018) in Federal Institute of Education, Science and Technology of Ceará. He is currently an M.Sc. candidate in Computer Science at Federal Institute of Education, Science and Technology of Ceará. He is interested in research in the areas of Embedded Systems, Computational Vision, Computational Intelligence and Pattern Recognition.

Gustavo Pinheiro Correia is now a college student at Federal Institute of Education, Science and Technology of Ceará, Brazil. His main research interest is digital signal processing.